Ausgaben formatieren

- Formatieren mit format()
- System.out.printf()
- Die Formatter-Klasse
- Formatieren mit Masken
- Format – Klassen
- NumberFormat & DecimalFormat

Formatieren mit format()

- Klasse String mit statischer Methode format() ermöglicht es Zeichenketten zu formatieren.

```
String s = String.format( "Hallo %s. Es gab einen Anruf von %s.",
                          "Chris", "Joy" );
System.out.println( s );  // Hallo Chris. Es gab einen Anruf von Joy.
```

- Format-String:
  - auszugebende Zeichen,
  - Format-Spezifizierer (Formatierung des Arguments)
  - Vararg - Feld mit den Werten

Formatieren mit format()

Spezifizierer	Steht für...
%n	neue Zeile
%%	Prozentzeichen
%c	Unicode-Zeichen
%x	Hexadezimalschreibweise
%f	Fließkommazahl
%b	Boolean
%s	String
%d	Dezimalzahl
%t	Datum und Zeit
%e	wissenschaftliche Notation

Formatieren mit format()

- **Zeilenvorschub** abhängig vom Betriebssystem
- mit %n kommt man an das Zeilenvorschubzeichen
- der format()-Aufruf kommt mit einem Argument aus, und es lautet String.format("%n")

Formatieren mit format()

```
static String format(String format, Object... args)
```

- Formatierter String aus dem String und den Argumenten

```
static String format(Locale l, String format, Object... args)
```

- Formatierter String aus der gewünschten Sprache, dem String und den Argumenten

`java.util.Formatter`

- wird intern verwendet

Formatieren mit format()

```
public static void main(String[] args)
{
  for (int i = 5; i < 199; i += 37) {
    System.out.format("Aktueller Wert: %3d%n", i);
  }
}
```

```
Aktueller Wert:   5
Aktueller Wert:  42
Aktueller Wert:  79
Aktueller Wert: 116
Aktueller Wert: 153
Aktueller Wert: 190
```

System.out.printf()

- format()-Methoden auch unter printf() zugänglich
- printf() ist eine Weiterleitung zur Methode format()
- Bp.: Gib Zahlen von 0 bis 16 hexadezimal aus:

System.out.printf()

%n Zeilenvorschub

%% liefert das Prozentzeichen

**** maskiert in einem String den Backslash aus

%s liefert String. **%S** schreibt Ausgabe groß

%b schreibt Boolean. **%B** schreibt String groß

System.out.printf()

%c schreibt Zeichen. **%C** in Großbuchstaben

Ganzzahlige numerischen Ausgaben:
%d (Dezimal)
%x (Hexadezimal)
%o (Oktal)
%X schreibt die hexadezimalen Buchstaben groß

Fließkommazahlen: **%f** oder **%e (%E)**.
Standardpräzision sind sechs Nachkommastellen.

System.out.printf()

```
for (double x=-0.2; x<1; x+=0.1)
  System.out.printf ("%e %e %e\n",
    x, Math.sin(x), Math.cos(x));
```

```
-2.000000e-01 -1.986693e-01 9.800666e-01
-1.000000e-01 -9.983342e-02 9.950042e-01
0.000000e+00 0.000000e+00 1.000000e+00
1.000000e-01 9.983342e-02 9.950042e-01
2.000000e-01 1.986693e-01 9.800666e-01
3.000000e-01 2.955202e-01 9.553365e-01
4.000000e-01 3.894183e-01 9.210610e-01
5.000000e-01 4.794255e-01 8.775826e-01
6.000000e-01 5.646425e-01 8.253356e-01
7.000000e-01 6.442177e-01 7.648422e-01
8.000000e-01 7.173561e-01 6.967067e-01
```

System.out.printf()

```
for (double x=-0.2; x<1; x+=0.1)
  System.out.printf ("%f %f %f\n",
    x, Math.sin(x), Math.cos(x));
```

```
-0.200000 -0.198669 0.980067
-0.100000 -0.099833 0.995004
0.000000 0.000000 1.000000
0.100000 0.099833 0.995004
0.200000 0.198669 0.980067
0.300000 0.295520 0.955336
0.400000 0.389418 0.921061
0.500000 0.479426 0.877583
0.600000 0.564642 0.825336
0.700000 0.644218 0.764842
```


System.out.printf()

```
System.out.printf("Dies %seine%s\n",
                  "ist ", " Zeile!");
```

```
Dies ist eine Zeile!
```

Beachte:

- Mischung von Formatstring und den nachfolgenden auszugebenden Strings,
- Ausgabe der Leerzeichen in Formatstring oder auszugebendem String,
- kein Leerzeichen zwischen `"%s"` und `"eine"`,
- explizite Ausgabe des Zeilenendes.

System.out.printf()

- Format-Spezifizierer für Datumswerte

Symbol	Beschreibung
%tA, %ta	Vollständiger/abgekürzter Name des Wochentags
%tB, %tb	Vollständiger/abgekürzter Name des Monatsnamens
%tC	Zweistelliges Jahrhundert (00–99)
%te, %td	Monatstag numerisch ohne bzw. mit führenden Nullen (1–31 bzw. 01–31)
%tk, %tl	Stundenangabe bezogen auf 24 bzw. 12 Stunden (0–23, 1–12)
%tH, %tI	Zweistellige Stundenangabe bezogen auf 24 bzw. 12 Stunden (00–23, 01–12)
%tj	Tag des Jahres (001–366)
%tM	Zweistellige Minutenangabe (00–59)
%tm	Zweistellige Monatsangabe (in der Regel 01–12)
%tS	Zweistellige Sekundenangabe (00–59)

System.out.printf()

- Format-Spezifizierer für Datumswerte

Symbol	Beschreibung
%tY	Vierstellige Jahresangabe
%ty	Die letzten beiden Ziffern der Jahresangabe (00–99)
%tZ	Abgekürzte Zeitzone
%tZ	Zeitzone mit Verschiebung zur GMT
%tR	Stunden und Minuten in der Form %tH:%tM
%tT	Stunden/Minuten/Sekunden in der Form %tH:%tM:%tS
%tD	Datum in der Form %tm/%td/%ty
%tF	ISO-8601-Format %tY-%tm-%td
%tc	Komplettes Datum mit Zeit in der Form %ta %tb %td %tT %tZ %tY

System.out.printf()

- Format-Spezifizierer für Datumswerte
- Bezieht sich ein Formatelement auf das vorangehende Argument, so kann ein < gesetzt werden:

```
Calendar c1 = new GregorianCalendar( 1973, 2, 12 );
Calendar c2 = new GregorianCalendar( 1985, 8, 2 );
System.out.printf( "%te. %<tb %<ty, %2$te. %<tb %<ty%n",
                   c1,              c2 );     // 12. Mrz 73, 2. Sep 85
```


Die Formatter-Klasse

- Die Methoden format() und prinf() delegieren die Formattierung an die Klasse java.util.Formatter

```
public static String format( String format, Object ... args )
{
  return new Formatter().format( format, args ).toString();
}
```

- Formatter:
  - Übernimmt die Formatierung
  - Gibt die formatierten Ausgaben an ein Ziel weiter

Die Formatter-Klasse

```
StringBuilder sb = new StringBuilder();
for ( double d = 0; d <= 1; d += 0.1 )
{
  String s = String.format( "%.1f%n", d );
  sb.append( s );
}
System.out.println( sb );   // 0,1 0,2 ... 1,0
```

- Jeder Schleifendurchlauf: Aufbau neuer Formatter

```
StringBuilder sb = new StringBuilder();
for ( double d = 0; d <= 1; d += 0.1 )
{
  String s = new Formatter().format( "%.1f%n", d ).toString();
  sb.append( s );
}
System.out.println( sb );   // 0,1 0,2 ... 1,0
```


Die Formatter-Klasse

Angabe des Ziels bedeutet effizientere Programmierung

```
StringBuilder sb = new StringBuilder();
Formatter formatter = new Formatter( sb );

for ( double d = 0; d <= 1; d += 0.1 )
  formatter.format( "%.1f%n", d );

System.out.println( formatter );   // 0,1 0,2 ... 1,0
```

Die Formatter-Klasse

- Formatspezifizierer %s kann auf jedem Argumenttyp angewendet werden, durch die Varargs werden auch primitive Elemente zu Wrapper-Objekten mit einer toString()-Methode
- Implementiert die Klasse die Schnittstelle java.util.Formattable, so ruft der Formatter nicht die toString()-Methode auf, sondern formatTo(Formatter formatter, int flags, int width, int precision).
- Die API-Dokumentation liefert ein Beispiel

Formatieren mit Masken

- Originalstring wird in einen Ausgabestring konvertiert und dabei neue Zeichen zur Ausgabe eingefügt mit Hilfe die Java-API Klasse javax.swing.text.MaskFormatter

```
MaskFormatter mf = new MaskFormatter( "**-**-****" );
mf.setValueContainsLiteralCharacters( false );
String valueToString = mf.valueToString( "12031973" );
System.out.println( valueToString );       // 12-03-1973
Object stringToValue = mf.stringToValue( valueToString );
System.out.println( stringToValue );       // 12031973
```

Formatieren mit Masken

- Musterplatzhalter

	Steht für
*	jedes Zeichen
#	eine Zahl, wie `Character.isDigit()` sie testet
?	ein Zeichen nach `Character.isLetter()`
A	ein Zeichen oder eine Ziffer, also `Character.isLetter()` oder `Character.isDigit()`
U	ein Zeichen nach `Character.isLetter()`, aber konvertiert in einen Großbuchstaben
L	ein Zeichen nach `Character.isLetter()`, aber konvertiert in einen Kleinbuchstaben
H	ein Hexadezimalzeichen (0–9, a–f oder A–F)
'	einen ausmaskierten und nicht interpretierten Bereich

Format - Klassen

`DateFormat`: Formatieren von Datums-/Zeitwerten
`NumberFormat`: Formatieren von Zahlen
`MessageFormat`: Formatieren für allgemeine Programmmeldungen

- Gründe für den Einsatz der Format-Klassen:
  - In String gibt es eine format()-Methode, aber keine parseXXX()-Methode
  - statischen getXXXInstance()-Methoden liefern vordefinierte Format-Objekte, die übliche Standardausgaben erledigen, etwa gerundete Ganzzahlen, Prozente oder unterschiedlich genaue Datums-/Zeitangaben

Format - Klassen

Ergebnis	Formatiert mit
02.09.2005	`DateFormat.getDateInstance().format( new Date() )`
15:25:16	`DateFormat.getTimeInstance().format( new Date() )`
02.09.2005 15:25:16	`DateFormat.getDateTimeInstance().format( new Date() )`
12.345,679	`NumberFormat.getInstance().format( 12345.6789 )`
12.345,68 €	`NumberFormat.getCurrencyInstance().format( 12345.6789 )`
12 %	`NumberFormat.getPercentInstance().format( 0.123 )`

DateFormat.getDateInstance().format(date) berücksichtigt korrekt je nach Land die Reihenfolge von Tag, Monat und Jahr und das Trennzeichen.

NumberFormat & DecimalFormat

```
System.out.println( NumberFormat.getInstance().format( 2E30 ) );
System.out.println( NumberFormat.getInstance().format( 2E-30 ) );
```

```
2.000.000.000.000.000.000.000.000.000.000
0
```

NumberFormat & DecimalFormat

```
abstract class java.text.NumberFormat
extends Format
```

`static NumberFormat getNumberInstance()`

Einfacher Formatierer für Zahlen.

`static NumberFormat getIntegerInstance()`

Schneidet Nachkommateil ab und rundet

`static NumberFormat getPercentInstance()`

Fließkommazahlen als Prozentzahl

`static NumberFormat getCurrencyInstance()`

Formatierer für Währungen

NumberFormat & DecimalFormat

DecimalFormat ist eine Unterklasse von NumberFormat und ermöglicht individuellere Anpassungen.

```
double d = 12345.67890;
DecimalFormat df = new DecimalFormat( "###,##0.00" );
System.out.println( df.format(d) );          // 12.345,68
```

NumberFormat & DecimalFormat

```
NumberFormat frmt1 = DecimalFormat.getCurrencyInstance( Locale.FRANCE );
System.out.println( frmt1.format( 12345.6789 ) );          // 12 345,68 €
```

```
NumberFormat frmt = DecimalFormat.getCurrencyInstance( Locale.ENGLISH );
frmt.setCurrency( Currency.getInstance( "EUR" ) );
System.out.println( frmt.format( 12345.6789 ) );  // EUR12,345.68
```


NumberFormat & DecimalFormat

```
for ( Currency currency : Currency.getAvailableCurrencies() )
{
  System.out.printf( "%s, %s, %s (%s)%n",
                     currency.getCurrencyCode(),
                     currency.getSymbol(),
                     currency.getDisplayName(),
                     currency.getDisplayName(Locale.ENGLISH) );
```

```
EGP, EGP, Ägyptisches Pfund (Egyptian Pound)
IQD, IQD, Irak Dinar (Iraqi Dinar)
GHS, GHS, Ghana Cedi (Ghana Cedi)
AFN, AFN, Afghani (Afghani)
MUR, MUR, Mauritius Rupie (Mauritius Rupee)
SGD, SGD, Singapur Dollar (Singapore Dollar)
...
```